JOURNAL OF COMPUTER AND SYSTEM SCIENCES 9, 308–316 (1974)

An Observation on Time-Storage Trade Off

STEPHEN A. COOK

Department of Computer Science, University of Toronto, Toronto, Ontario M5S 1A7, Canada

Received August 30, 1973

There are two main results proved here. The first states that a certain set SP of strings (those coding "solvable path systems") has tape complexity $(\log n)^2$ iff every set in $\mathcal{P}$ (i.e., of deterministic polynomial time complexity) has tape complexity $(\log n)^2$. The second result gives evidence that SP does not have tape complexity $(\log n)^k$ for any k.

Recently there have been several attempts to prove that every set of strings in $\mathcal{P}$ (i.e., recognizable in deterministic polynomial time; see [1, 2]) can be recognized in deterministic storage $(\log n)^2$. The method used in the attempts was based on that of [3], in which it is shown that every context free language can be accepted in storage $(\log n)^2$. Our thesis in the present paper is that these attempts must fail. We define a specific set SP of strings which is clearly in $\mathcal{P}$, but in a certain well-defined sense cannot be recognized in storage $(\log n)^2$ using the techniques in [3]. We conjecture that no Turing machine recognizes SP within storage $(\log n)^2$, and show that if this conjecture is false, then in fact every member of $\mathcal{P}$ can be recognized within storage $(\log n)^2$.

Our use of the terms *tape complexity* and *time complexity* is the standard one defined in [4]. Thus $\mathcal{P}$ is the class of all sets A of strings such that for some c, k, A has time complexity cn^k.

The notion of path systems was introduced in [5]. We repeat the definition here.

DEFINITION. A *path system* is a quadruple $\mathcal{S} = \langle X, R, S, T\rangle$, where X is a finite set (of nodes), R is a three place relation on X (the *incidence* relation), $S \subseteq X$ (S is the set of *source* nodes), and $T \subseteq X$ (T is the set of *terminal* nodes).

The *admissible* nodes of $\mathcal{S}$ are the least set A such that $T \subseteq A$ and such that if $y, z \in A$, and $R(x, y, z)$ holds, then $x \in A$. We say $\mathcal{S}$ is *solvable* iff at least one admissible node is a source node (i.e., member of S).

For a natural example, let X be the set of all formulas of length not exceeding l in some formal system (such as clauses in a resolution system for propositional calculus). Let the incidence relation $R(A, B, C)$ hold iff the formula A follows from

formulas B and C by a rule of inference (such as resolution or modus ponens). Let $T \subseteq X$ be a set of axioms, and let $S = \{D\}$, where $D \in X$. (D might be the empty clause for the case of resolution.) Then the admissible nodes of the path system $\mathscr{S} = \langle X, R, S, T \rangle$ are just those formulas which can be derived from the axioms T, and $\mathscr{S}$ is solvable iff D can be derived from T. In the case of resolution, $\mathscr{S}$ is solvable iff the clauses T are inconsistent.

The definition of solvable for a path system can be given alternatively as follows. Starting with a node $s \in S$, we try to form a directed graph by finding nodes $x_1, x_2 \in X$ so $R(s, x_1, x_2)$ holds. Then, we let s, x_1, x_2 be vertices of the graph, and let (s, x_1) and (s, x_2) be directed edges. Then we find x_3, x_4 and x_5, x_6 so $R(x_1, x_3, x_4)$ and $R(x_2, x_5, x_6)$ hold, and let (x_1, x_3), (x_1, x_4), (x_2, x_5), (x_2, x_6) be directed edges as shown in Fig. 1. We continue in this way (possibly letting a node have two or more edges coming in) until every directed path leads to a terminal node (member of T). This is possible iff $\mathscr{S}$ is solvable.

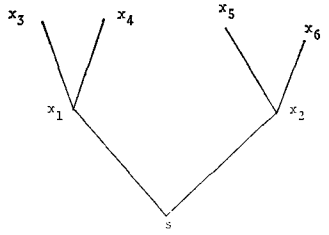

FIGURE 1

The structure formed in this process is not necessarily a tree as shown in Fig. 1, but in general it is an acyclic digraph. However, for some special path systems the structure described above can be aasumed to be a tree. An example of this comes from considering a context free grammar G in Chomsky normal form. In [5] it is shown how one can associate with each pair $\langle w, G \rangle$ a path system $S(w, G)$ which is solvable iff w is in the language of G, where w is a string on the alphabet of terminal symbols of G. If w is in the language of G, then the structure shown in Fig. 1 showing solvability is essentially the parse tree of w. The result of Lewis, Stearns, and Hartmanis [3] stating that every context free grammar can be recognized in storage $(\log n)^2$ can be proved by showing the corresponding "tree-like" path systems can be checked for sovability in storage $(\log n)^2$ (see [5], and Corollaries 2 and 3 of Theorem 4 below). The argument giving storage $(\log n)^2$ breaks down if the path systems are not "tree-like," i.e., if Fig. 1 cannot be taken to be a tree (see Theorem 5 and the discussion following it). This seems to be the trap which has caught several researchers trying to show every language in $\mathscr{P}$ can be recognized in storage $(\log n)^2$.

Let us code a path system $\mathscr{S} = \langle X, R, S, T \rangle$ as a string $\phi(\mathscr{S})$ on the alphabet $\{0, 1, *\}$ by coding the nodes of X with strings on $\{0, 1\}$, and then listing the codes

for the members of X, followed by the triples of R and members of S and T, using $*$ appropriately as a separator. The code for the nodes should be efficient in the sense that if there are N nodes, then no node is assigned a string of length exceeding $\log_2 N + 1$. Thus the length of the code $\phi(\mathcal{S})$ is $O(N^3 \log N)$.

DEFINITION. SP is the set of all strings over $\langle 0, 1, * \rangle$ which code solvable path systems.

THEOREM 1. $SP \in \mathcal{P}$.

Proof. Suppose the string $\phi(\mathcal{S})$ codes the path system $\mathcal{S}$ of N nodes. The (codes for the) admissible nodes of $\mathcal{S}$ can be enumerated in at most n (= length of $\phi(\mathcal{S})$) stages, where the first stage consists of enumerating the members of T, and the $(i+1)$st stage consists of considering each pair (y, z) of nodes which have been previously enumerated and enumerating each x such that $R(x, y, z)$ holds, but x has not been previously enumerated. The time for each stage is certainly $O(n^3)$, so the entire computation time is $O(n^4)$.

CONJECTURE. *SP does not have either deterministic or nondeterministic tape complexity $(\log n)^k$, for any k.*

In the following theorem, all complexities refer to deterministic machines.

THEOREM 2. *If SP has tape complexity $(\log n)^k$, then for every function $T(n) \geqslant n$, every language of time complexity $T(n)$ has tape complexity $(\log T(n))^k$. In particular, every member of $\mathcal{P}$ would have tape complexity $(\log n)^k$.*

Proof. What we actually show is that every language of time complexity $T(n)$ is reducible in storage $\log T(n)$ to SP. (See [1] for a similar notion of reducibility.) We will not phrase the proof this way, however.

The proof of Theorem 2 uses a construction used in the proof of the main theorem of [6]. The part of that theorem which concerns us states that if L has time complexity $T(n) \geqslant n$, then L is accepted by some auxiliary pushdown machine with storage $\log(T(n))$. The idea of the proof is to associate a path system $\mathcal{S} = \mathcal{S}(w, Z)$ with each pair (w, Z), where Z is a special Turing machine accepting L and w is an input to Z. (The notion of "path system" is not actually identified in [6].) The path system is solvable iff Z accepts w, and it turns out that the solvability of $\mathcal{S}$ can be detected by an auxiliary pushdown machine within storage $\log T(n)$.

The special Turing machine Z which accepts L has only one tape (which serves as both an input and work tape), and the read/write head of Z moves in an easily predicted motion. At the beginning of the computation the head moves right n squares so that it scans one square to the right of the input w, and prints a symbol. Then it

moves left $n + 1$ squares, then right $n + 2$ squares, and so on. It is easy to see that Z can be designed to accept L within time $(T(n))^d$ for some d, if there is a multitape Turing machine which accepts L within time $T(n)$.

Now $\mathscr{S}(w, Z) = \langle X, R, S, T\rangle$, where $X = \{(t, q, s) \mid t$ is an integer between 0 and $(T(n))^d$, q is a state of Z, and s is a tape symbol of $Z\}$. Before defining R, S, T, we motivate the definition of X by saying that the admissible nodes of $\mathscr{S}$ will be just those triples (t, q, s) such that Z is in state q scanning a symbol s at time t of its computation with input w. Then $R(x, y, z)$ holds iff the triples y, z "yield" the triple x in the sense of [6, p. 10]. The idea is that (t, q, s), (t_1, q_1, s_1) yield (t_2, q_2, s_2) iff $t_2 = t + 1$ and (t_2, q_2, s_2) can be deduced admissible given that (t, q, s) and (t_1, q_1, s_1) are. Precisely, $R((t_2, q_2, s_2), (t, q, s), (t_1, q_1, s_1))$ holds iff $t_2 = t + 1$ and when Z is in state q scanning the symbol s, it next assumes the state q_2, and either (i) Z scans a square for the first time at time $t + 1$, in which case s_2 is the symbol originally occupied by that square at the start of the computation, or (ii) t_1 is the greatest integer less than $t + 1$ such that Z scans the same square at steps t_1 and $t + 1$, in which case s_2 is the symbol printed by Z when in state q_1 scanning the symbol s_1.

We complete the definition of $\mathscr{S}$ by defining $S = \{(t, q, s) \mid q$ is an accepting state of $Z\}$ and $T = \{(0, q_0, w_1)\}$ where q_0 is the initial state of Z and w_1 is the first symbol of w (i.e., the symbol Z scans at the start of its computation).

It is easy to check that (t, q, s) is admissible iff Z is in state q scanning the symbol s at time t, as claimed above. Hence Z accepts w iff $\mathscr{S}(w, Z)$ is solvable.

Suppose now that SP is accepted within storage $(\log n)^k$ by a Turing machine M_{SP}. Then a Turing machine M_L to accept L within storage $(\log(T(n)))^k$ could proceed on input w by simulating M_{SP} on input $\phi(\mathscr{S}(w, Z))$ (a string coding $\mathscr{S}(w, Z)$). But there are two difficulties with this procedure. First, it may not be possible to calculate the function $T(n)$ in storage $(\log T(n))^k$; second, $\phi(\mathscr{S}(w, Z))$ is too long to write on a work tape within storage $(\log T(n))^k$. The first difficulty is overcome by using a parameter t for $T(n)$, and repeating the entire computation for every value of t: $t = 1, 2, \ldots$. Thus M_L will only halt if for some value of t the calculated $\phi(w, Z)$ is solvable. That is, M_L halts iff Z accepts w.

The second difficulty is overcome by not writing the entire string $\phi(\mathscr{S}(w, Z))$ at once. In fact, it suffices for M_L to be able to calculate the ith symbol of $\phi(\mathscr{S}(w, Z))$, given i. This calculation can be done in storage $O(\log(T(n))^d) = O(\log T(n))$, i.e., a constant times the storage needed to write a triple (t, q, s). (We assume integers such as i and t are written in binary notation.) This follows from the following lemma.

LEMMA 1. *The relation R of the path system $\mathscr{S}(w, Z)$ can be computed (i.e., verified) within storage $O(\log T(n))$ by the machine M_L, where n is the length of w.*

Proof. We assume the input triples $((t_2, q_2, s_2), (t, q, s), (t_1, q_1, s_1))$ are available on a work tape of M_L, with t, t_1, t_2 all written in binary notation. To check whether

these triples satisfy the relation R, M_L makes use of the procedure head(τ) given below which calculates the head position H of the Turing machine Z after τ steps of its computation (on input w). (See the description of the head motion of Z above). Here L and R are variables for the squares where Z's head should reverse direction. The variable D tells the current direction of Z's head, with $D = 1$ meaning right, and $D = 0$ meaning left.

procedure head(τ)
$H \leftarrow 1$; $L \leftarrow 1$; $R \leftarrow n$; $D \leftarrow 1$;
while $\tau > 0$ *do begin*
  if $D = 1$ *then begin* $H \leftarrow H + 1$;
    if $H = R + 1$ *then begin* $R \leftarrow R + 1$; $D \leftarrow 0$ *end else*
  end
  else begin $H \leftarrow H - 1$;
    if $H = L - 1$ *then begin* $L \leftarrow L - 1$; $D \leftarrow 1$ *end else*
  end;
  $\tau \leftarrow \tau - 1$
end.

Upon termination of the procedure, the value of H will be the number of the square scanned by Z's head after τ steps of Z's computation (here the original value of τ is referred to). Referring to the definition of R, we note that Z scans a square for the first time at time $t + 1$ iff $t + 1 \leqslant n$, or when the procedure head($t + 1$) is called, upon termination $H = R$ or $H = L$. Similarly, the condition "t_1 is the greatest integer less than $t + 1$ such that Z scans the same square at steps t_1 and $t + 1$" can be checked by calling head(τ) for successive values of τ from t_1 to $t + 1$. No variable of the procedure ever exceeds $\max(\tau, n)$ during execution of a call head(τ). Therefore, since all values can be stored in binary notation, and since in order to check the relation R, head(τ) need never be called for $\tau > T(n)$, we see that M_L can check the relation R within storage $O(\log T(n))$, as required by the lemma.

To complete the proof of Theorem 2, we note that the total storage required by M_L is $O(\log T(n))$ plus the storage required by M_{SP}. The latter by assumption, is $(\log N)^k$, where N is the length of $\phi(\mathscr{S}(w, Z))$, i.e., $N = O((T(n))^e)$ for some e. Thus the storage required by M_L is $O((\log T(n))^k)$, as required by the theorem.

COROLLARY 1. *If SP has tape complexity* $(\log n)^2$, *then every member of* $\mathscr{P}$ *has tape complexity* $(\log n)^2$.

We now describe a special purpose machine for recognizing solvable path systems. The machine can be thought of as playing a game similar to the one described by Paterson and Hewitt [7] or by Strong and Walker [8] to carry out a computation of

a program schema. The machine bears similarities to the maze recognizing automaton of Savitch [9].

DEFINITION. The *marking machine* M_0 operates on an input path system $\mathcal{S}$ as follows. A *configuration* C of $\mathcal{S}$ is simply a subset of the nodes of $\mathcal{S}$ (C is the set of *marked* nodes). The initial configuration is the empty set. In one move, M_0 changes a configuration C into C' (nondeterministically) by any one of three ways: first, marking a terminal node $t \in T$ (i.e., $C' = C \cup \{t\}$); second, if y, z are marked nodes and x is a node such that either $R(x, y, z)$ or $R(x, z, y)$, then M_0 removes the marker from y and places it on x (i.e., $C' = C \cup \{x\} - \{y\}$); and third, M_0 may remove the marker from any marked node. A configuration C is *accepting* iff some marked node is a source node (i.e., $C \cap S \neq \phi$). We say M_0 *accepts* $\mathcal{S}$ iff there is a sequence $C_0, C_1, \ldots, C_m$ of configurations of $\mathcal{S}$ representing legal moves in which C_0 is empty and C_m is accepting. M_0 *accepts* $\mathcal{S}$ using K *markers* iff the cardinality of no C_i in the accepting sequence exceeds K.

It is clear from the above definitions that a node of $\mathcal{S}$ is admissible iff some sequence of moves causes it to be marked. Hence M_0 accepts $\mathcal{S}$ iff $\mathcal{S}$ is solvable.

DEFINITION. $\mathcal{S} = \langle X, R, S, T \rangle$ is *tree solvable* iff there is a binary rooted tree whose nodes are in X, whose leaves are in T, whose root is in S, and if the node x has son nodes y, z, then $R(x, y, z)$ holds or $R(x, z, y)$ holds.

It is clear that every tree solvable path system is solvable, since all nodes on the tree must be admissible, including the root $r \in S$.

THEOREM 3. *If $\mathcal{S}$ is a tree solvable path system with N nodes, then M_0 accepts $\mathcal{S}$ using* $[\log_2 N] + 1$ *markers.*

We prove by induction on l the slightly stronger statement that if the tree showing $\mathcal{S}$ is solvable and has l leaves, then M_0 can mark the root using $[\log_2 l] + 1$ markers. For $l = 1$ this assertion is clear. Suppose $l > 1$. Let the root of the tree be r, and the two sons of r be the nodes y and z, and suppose at least half of the leaves of the tree lie below z. By the induction hypothesis, there is a sequence of moves using at most $[\log_2 l] + 1$ markers which causes z to be marked. Since the node y has at most $l/2$ leaves below it, and since $[\log_2(l/2)] + 1 = [\log_2 l]$, the induction hypothesis also implies that the node y can be marked in a sequence of moves using at most $[\log_2 l]$ markers, so it is possible to keep one marker on z and cause y to be marked at the same time and never use more than $[\log_2 l] + 1$ markers. Finally, the root r can be marked in one move (since there are markers on y and z, and $R(r, y, z)$ or $R(r, z, y)$ holds).

DEFINITION. A Turing machine Z with input alphabet $\{0, 1, *\}$ is *sound* iff every string accepted by Z codes a solvable path system.

In the following theorem, we say $f(n)$ is *constructable* if there is a Turing machine which operates within storage $f(n)$ and upon receiving an input string of 0's of length n, it can mark a string of length $f(n)$ on its storage tape.

THEOREM 4. *Let $f(n)$ be a constructable nondecreasing function on the natural numbers. Then there is a sound, nondeterministic, off-line Turing machine Z of tape complexity $f(n) \log n$ which accepts every code for every path system $\mathscr{S}$ which M_0 accepts within $f(N)$ markers, where N is the number of nodes of $\mathscr{S}$.*

Proof. Z simply simulates M_0 by keeping a list of the marked nodes. We note that if n is the length of the input string (coding $\mathscr{S}$), then the code for each node has at most $[\log_2 n] + 1$ bits, and $N \leqslant n$. Hence the bound $f(n) \log n$ for storage suffices. Since $f(n)$ is constructable, so is $f(n) \log n$, and Z can be fixed to halt if the storage ever threatens to exceed $f(n) \log n$.

COROLLARY 2. *If $f(n) = [\log_2 n] + 1$, then the Z in the theorem accepts every coded tree solvable system within storage* $(\log n)^2$.

This is immediate from Theorem 3.

COROLLARY 3. *Every context-free language has nondeterministic tape complexity* $(\log n)^2$.

This follows because every context-free grammar G in Chomsky normal form and string of terminals w can be associated with a path system $\mathscr{S} = \mathscr{S}(w, G)$ such that $\mathscr{S}$ is tree-solvable iff w is generated by G, and $\mathscr{S}$ is not solvable if w is not generated by G. Further, $\mathscr{S}$ has $O(n^2)$ nodes, where n is the length of w. Finally, the ith symbol of a code for $\mathscr{S}(w, G)$ can be easily calculated by a Turing machine with input w within storage $O((\log n)^2)$. $\mathscr{S}(w, G)$ is described in [5].

Of course the Turing machines in the above corollaries can be made deterministic, as indicated in [3, 5]. In this paper we are more interested in what *cannot* be recognized in (nondeterministic) storage $(\log n)^2$.

THEOREM 5. *For each m there is a solvable path system $\mathscr{S}_m$ with $m(m+1)/2$ nodes such that M_0 cannot accept $\mathscr{S}_m$ using fewer than m markers.*

Proof. $\mathscr{S}_m$ can be represented by a pyramid of m rows, whose bottom row has m nodes, the next row has $m - 1$ nodes, and so on until the top row has a single node. The top node r is the only source node ($S = \{r\}$), and the terminal nodes T comprise the bottom row. The incidence relation $R(x, y, z)$ holds iff y and z are the two "son" nodes of x. The system $\mathscr{S}_5$ is shown in Fig. 2.

The argument that M_0 requires m markers to accept $\mathscr{S}_m$ is similar to an argument used in [7] for a tree. Let us agree that a *path* of $\mathscr{S}_m$ is a sequence of m nodes $x_1, \ldots, x_m$ connecting the top node with some terminal (bottom node) (so x_{i+1} lies immediately

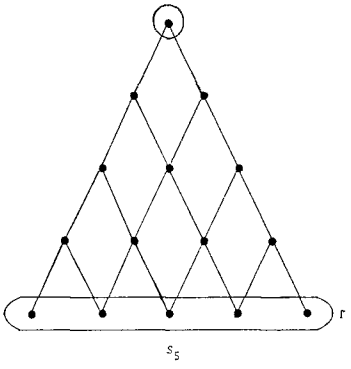

FIGURE 2

below x_i, all i). Consider an accepting sequence $C_0, C_1, \ldots, C_t$ of configurations for $\mathscr{S}_m$. Initially every path is free of markers (since C_0 is empty), but after C_t, every path hits a marker (namely the marker on the source node). Let $C_r, r < t$, be the last configuration such that at least one path p does not hit a marker. It is not hard to see that the move to configuration C_{r+1} involves placing a marker on the terminal node at the base of p. This is because any other move to a node x above marked nodes y and z could not block p unless p were already blocked by markers on y or z. From each node x_i of p, there is a unique path p_i which agrees with p from the source down to x_i, and then diverges from p_i and continues to a terminal node at the bottom in such a way that the nodes on this last segment of p_i from p to the terminal node lie on a straight line. An example of possible paths p_2 and p_3 for $\mathscr{S}_5$ is shown in Fig. 3.

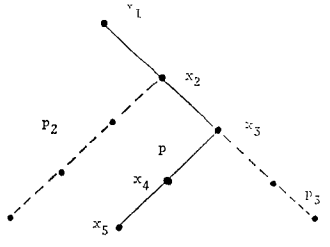

FIGURE 3

The path p_m coincides with p. Since configuration C_{r+1} has all paths blocked with markers, and no two of the m paths $p_1, p_2, \ldots, p_m$ are blocked with the same marker, it follows that C_{r+1} has at least m markers. The theorem follows.

The point of Theorem 5 is that the method which is formalized by the machine M_0 for showing a path system is solvable, would require storage $n^{1/2}$ at least if used on a Turing machine, since the machine would require at least one square to keep track of every marker used. Thus in one sense at least, the method used for recognizing

context-free languages within storage $(\log n)^2$ would require storage $n^{1/7}$ at least for the set SP, even if the machine were allowed to be nondeterministic. This is because an input string of length n can code a path system of $(n/\log n)^{1/3}$ nodes. (Recall that the triples satisfying the incidence relation R must be listed explicitly).

I hope this paper will stimulate research toward proving (or disproving) the conjecture stated earlier. A possible next step is to define a class of "marking machines" like M_0, only more general, which can accept solvable path systems, and prove that they still require more than $\log N$ markers to accept a solvable path system of N nodes.

REFERENCES

1. S. A. COOK, The complexity of theorem-proving procedures, "Proceedings of Third Annual ACM Symposium on Theory of Computing," pp. 151–158, May 1971.
2. R. M. KARP, Reducibilities among combinatorial problems, *in* "Complexity of Computer Computations" (R. E. Miller and J. W. Thatcher, Eds.), pp. 85–103, Plenum Press, New York, 1972.
3. P. M. LEWIS II, R. E. STEARNS, AND J. HARTMANIS, Memory Bounds for the Recognition of Context-Free and Context-Sensitive Languages, "IEEE Conference Record on 1965 Symposium on Switching Circuit Theory and Logical Design," Institute of Electrical and Electronics Engineers, New York.
4. J. E. HOPCROFT AND J. D. ULLMAN, "Formal Languages and Their Relation to Automata," Addison–Wesley, Reading, MA, 1969.
5. S. A. COOK, Path systems and language recognition, "Proceedings Second Annual ACM Symposium on the Theory of Computation," pp. 70–72, Association for Computing Machinery, New York, 1970.
6. S. A. COOK, Characterizations of Pushdown machines in terms of time-bounded computers, *J. Assoc. Comput. Mach.* **18** (1971), 4–18.
7. M. S. PATERSON AND C. E. HEWITT, Comparative schematology, "Record of Project MAC Conference on Concurrent Systems and Parallel Computations (June, 1970)," pp. 119–128, ACM, New Jersey, December 1970.
8. S. A. WALKER AND H. R. STRONG, Characterizations of Flow-chartable Recursions, Short Version, "Proceedings of Fourth-Annual ACM Symposium on Theory of Computing, pp. 18–34, May 1972.
9. W. J. SAVITCH, Maze recognizing automata and nondeterministic tape complexity, *J. Comput. System Sci.* **7** (1973), 389–403.